第3章

程序控制结构与数组

本章学习目标

- 掌握分支语句的使用方法。
- 掌握循环控制语句的使用方法。
- 掌握关键字 break 和 continue 的使用方法。
- 掌握一维数组和二维数组的使用方法。

通常情形下，程序代码按照从上往下顺序地执行。但在解决一些实际问题时，需先对一些条件进行判断，然后再决定执行哪段代码，或者当满足一定的条件时重复地执行某种操作，即要对程序运行进行控制，这就是程序控制结构。此外，当程序需要处理较多数据时，仅通过定义变量难以完成，通常需要使用数组来存储数据并对数据进行各种处理。本章主要介绍分支和循环两种程序控制结构，以及数组数据类型。

3.1 分支语句

分支语句分为 if 结构和 switch 结构两类。if 结构具有三种形式，分别是单分支 if 语句、双分支 if…else 语句、多分支 if…else if 语句，下面将逐一介绍。

3.1.1 if 语句

if 语句是单分支结构。它的作用是判断某个条件，如果条件成立，则执行某个操作。其语法格式如下。

```
if(判断条件)
{
    执行语句
}
```

上述语法格式中，判断条件是一个布尔值。当判断条件为 true 时，就会执行 if 后面{}中的执行语句。if 语句的执行流程如图 3.1 所示。

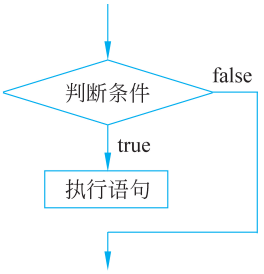

图 3.1　if 语句的执行流程

【例 3-1】　if 语句实例。


ex3_1

```
0001 public class Example3_1
0002 {
0003     public static void main(String[] args)
0004     {
0005         int score = 80;
0006         if (score >= 60)
0007         {
0008             System.out.println("分数为" + score + ",成绩合格。");
0009         }
0010     }
0011 }
```

【运行结果】

程序运行结果如图 3.2 所示。

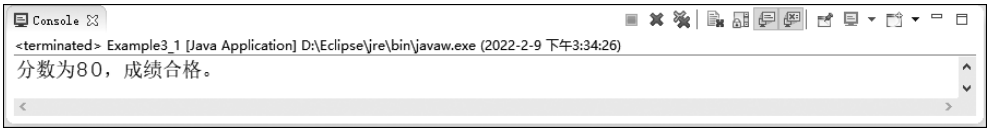

图 3.2　if 语句实例输出

【程序说明】

程序第 5 行定义了 int 类型的变量 score，表示某门课程成绩，并赋值为 80 分；第 6 行在 if 语句中设置判断条件为成绩大于或等于 60 分，即 score＞＝60，显然条件是成立的，将会选择执行 if 后面{}中的执行语句，从而输出“成绩合格”的信息。

特别地，当 if 后面{}中的执行语句只有一条时，可以省略{}。

3.1.2　if…else 语句

if…else 语句是双分支结构。它的作用是判断某个条件，如果条件成立，则执行某个操作，否则将执行另外某个操作。其语法格式如下。

```
if(判断条件)
{
    执行语句 1
}
else
{
    执行语句 2
}
```

上述语法格式中,判断条件是一个布尔值。当判断条件为 true 时,执行 if 后面{}中的执行语句 1,否则执行 else 后面{}中的执行语句 2。if…else 语句的执行流程如图 3.3 所示。

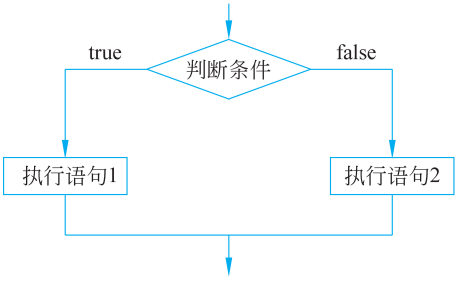

图 3.3 if…else 语句的执行流程


ex3_2

【例 3-2】 if…else 语句实例。

```
0001 public class Example3_2
0002 {
0003     public static void main(String[] args)
0004     {
0005         int score = 50;
0006         if(score >= 60)
0007         {
0008             System.out.println("分数为" + score + ",成绩合格。");
0009         }
0010         else
0011         {
0012             System.out.println("分数为" + score + ",成绩不合格。");
0013         }
0014     }
0015 }
```

【运行结果】

程序运行结果如图 3.4 所示。

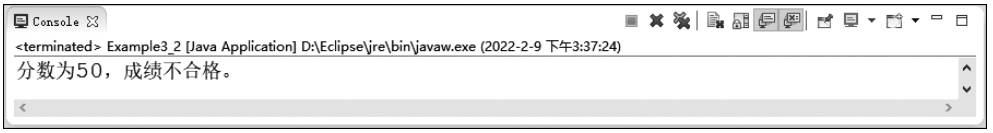

图 3.4　if…else 语句实例输出

【程序说明】

程序第 5 行定义了 int 类型的变量 score，表示某门课程成绩，并赋值为 50；在 if…else 语句中设置判断条件为成绩大于或等于 60 分，即 score＞＝60，显然条件是不成立的，那么会选择执行 else 后面{}中的执行语句，从而输出“分数为 50，成绩不合格”的信息。

【例 3-3】　判断奇偶性实例。


ex3_3

```
0001 import java.util.Scanner;
0002 public class Example3_3
0003 {
0004     public static void main(String[] args)
0005     {
0006         Scanner scan = new Scanner(System.in);
0007         int x = scan.nextInt();
0008         if(x % 2 == 0)
0009             System.out.println(x + "是偶数");
0010         else
0011             System.out.println(x + "是奇数");
0012         scan.close();
0013     }
0014 }
```

【运行结果】

程序运行结果如图 3.5 所示。

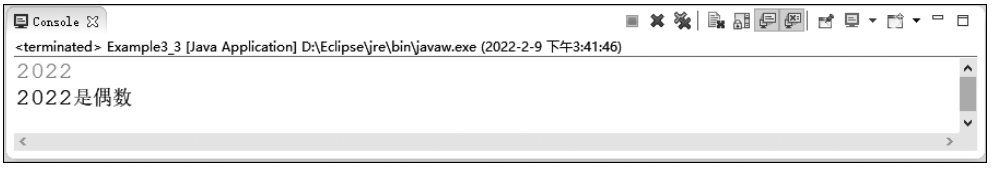

图 3.5　判断奇偶性实例输出

【程序说明】

程序第 7 行从键盘读入一个整型值存储在变量 x 中，然后通过 if…else 语句判断其奇偶性，判断条件设置为 x%2＝＝0，程序运行并输入“2022”后，显然条件满足，所以执行 if 后面的语句块，即第 9 行，输出“2022 是偶数”。

3.1.3 if…else if 语句

if…else if 语句是多分支结构。它的作用是先判断第一个条件,如果该条件成立,则执行第一段代码;否则将判断第二个条件,如果该条件成立,则执行第二段代码;否则继续判断下一个条件,以此类推。其语法格式如下。

```
if(判断条件 1)
{
    执行语句 1
}
else if(判断条件 2)
{
    执行语句 2
}
…
else if(判断条件 n)
{
    执行语句 n
}
else
{
    执行语句 n+1
}
```

上述语法格式中,判断条件均为布尔值。当判断条件 1 为 true 时,执行 if 后面{}中的执行语句 1;否则继续判断条件 2,当判断条件 2 为 true 时,执行 else if 后面{}中的执行语句 2,以此类推。当判断条件 1～判断条件 n 均为 false 时,则执行 else 后面{}中的执行语句 n+1,if…else if 语句的执行流程如图 3.6 所示。

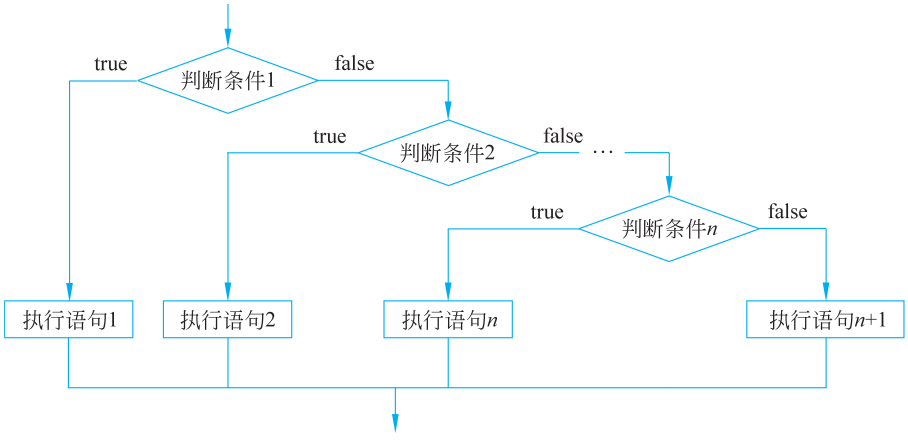

图 3.6 if…else if 语句的执行流程

【例 3-4】　if…else if 语句实例。

ex3_4

```
0001 public class Example3_4
0002 {
0003     public static void main(String[] args)
0004     {
0005         /* 本程序的功能是根据课程成绩划分等级,设分数与等级的对应关系:
0006         85~100,A 等;75~84,B 等;60~74,C 等;0~59,D 等 */
0007         int score = 83;
0008         if(score >= 85)
0009             System.out.println("成绩为 A 等");
0010         else if(score >= 75)
0011             System.out.println("成绩为 B 等");
0012         else if(score >= 60)
0013             System.out.println("成绩为 C 等");
0014         else
0015             System.out.println("成绩为 D 等");
0016     }
0017 }
```

【运行结果】

程序运行结果如图 3.7 所示。

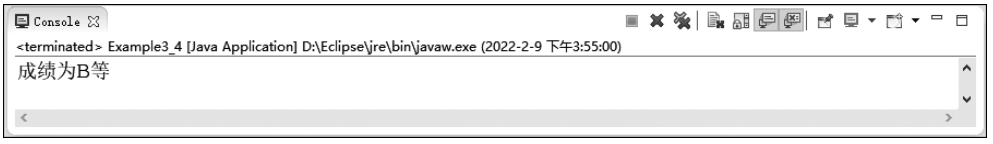

图 3.7　if…else if 语句实例输出

【程序说明】

程序第 7 行定义了一个 score 变量表示成绩;程序第 8～15 行使用 if…else if 语句根据成绩分为 A、B、C、D 共 4 个等级,设置相应条件及执行语句,共 4 个分支。score 值为 83,和第一个 else if 的条件匹配,因此输出“成绩为 B 等”。

3.1.4　switch 语句

switch 语句是一种常用的多分支结构语句,它由一个 switch 测试表达式和多个 case 测试项组成。与 if 语句中的判断条件不同,switch 测试表达式的类型只能是 byte、short、char、int、enum 枚举以及 String 类型,而非 boolean 类型。其语法格式如下。

```
switch(测试表达式)
{
    case  测试项 1:
        执行语句 1;
```

```
        break;
    case  测试项 2:
        执行语句 2;
        break;
…
    case  测试项 n:
        执行语句 n;
        break;
    default:
        执行语句 n+1;
}
```

switch 语句执行的流程是,将测试表达式的值逐个与 case 后的测试项进行匹配,如果某个 case 项匹配成功了,则执行该 case 项后面的所有语句,直到 switch 结构结束。因此,为了只执行匹配的那个 case 语句,在 case 语句后面通常会使用 break 关键字。如果没有任何一个 case 项匹配成功,那么将执行 default 后的语句。


ex3_5

【例 3-5】 switch 语句实例。

```
0001 public class Example3_5
0002 {
0003    public static void main(String[] args)
0004    {
0005        int month = 10;
0006        switch(month)
0007        {
0008            case 1:
0009            case 2:
0010            case 12:
0011                System.out.println("当前为" + month + "月,是冬季");
0012                break;
0013            case 3:
0014            case 4:
0015            case 5:
0016                System.out.println("当前为" + month + "月,是春季");
0017                break;
0018            case 6:
0019            case 7:
0020            case 8:
0021                System.out.println("当前为" + month + "月,是夏季");
0022                break;
0023            case 9:
0024            case 10:
0025            case 11:
```

```
0026                System.out.println("当前为" + month + "月,是秋季");
0027                break;
0028            default:
0029                System.out.println("月份应该在 1~12 之间");
0030        }
0031    }
0032 }
```

【运行结果】

程序运行结果如图 3.8 所示。

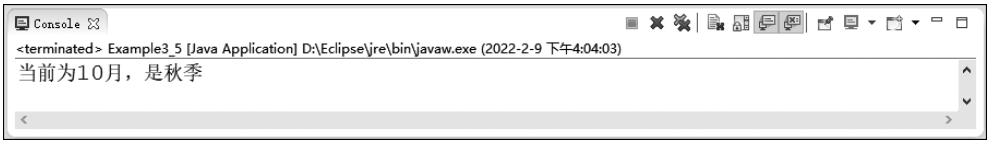

图 3.8　switch 语句实例输出

【程序说明】

程序将变量 month 的值与各个 case 后的值进行匹配，执行到第 24 行匹配成功，并继续往下执行，执行到第 27 行的 break 语句跳出 switch 结构。读者可以尝试将第 27 行代码注释后再运行，来更好地理解 switch 结构的执行流程。此时将输出两行信息“秋季”“月份应该在 1～12 之间”，因为执行到第 24 行匹配后，将继续往下执行，直到 switch 结构结束，中间未遇到跳出语句 break。

3.2　循环控制语句

除了分支结构之外，另一种重要的程序控制结构是循环结构，通过它可以实现一些操作的重复执行。本节将介绍循环控制语句的三种主要形式，分为 while 循环、do … while 循环和 for 循环。

3.2.1　while 语句

while 语句在形式上与 3.1.1 节中的 if 语句相似，都是根据条件判断来决定是否执行{}中的语句。区别在于，while 语句会反复进行循环条件测试，只要循环条件成立，就继续执行{}中语句，如果循环条件不成立，则结束 while 循环。其语法格式如下。

```
while (循环条件)
{
    循环体
}
```

上述语法格式中，循环条件是一个布尔值，花括号{}之间的执行语句称为循环体，其

执行流程如图 3.9 所示。首先,判断循环条件,如果条件成立,则执行循环体,然后继续判断条件;如果条件不成立,则终止循环结构。

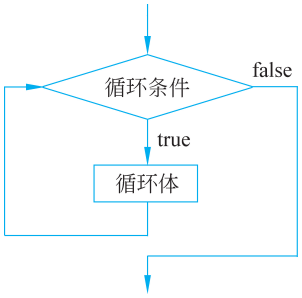

图 3.9 while 语句的执行流程


ex3_6

【例 3-6】 while 语句实例。

```
0001 public class Example3_6
0002 {
0003     public static void main(String[] args)
0004     {
0005         int sum = 0;
0006         int i = 1;
0007         while (i <= 1000)
0008         {
0009             sum += i;
0010             i++;
0011         }
0012         System.out.println("1~1000 的和为: " + sum);
0013     }
0014 }
```

【运行结果】

程序运行结果如图 3.10 所示。

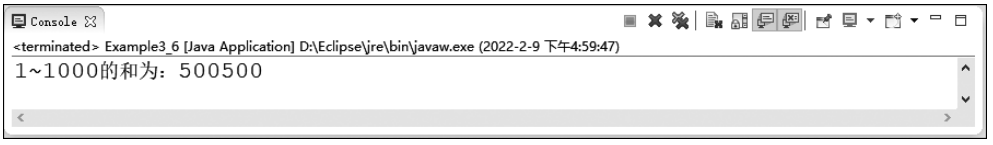

图 3.10 while 语句实例输出

【程序说明】

本程序的功能是求 1+2+3+…+1000 的和。在使用循环结构时,首先需在循环结构之前完成相关变量初始化,即程序第 5～6 行;其次确定循环条件,控制循环的进行和终止,即程序第 7 行;最后应分析循环重复执行的操作,确定循环体的内容,如例 3-6 中重复执行的操作即累加求和,即程序第 9～10 行。

【例 3-7】　判断自然数位数实例。


```
0001 public class Example3_7
0002 {
0003     public static void main(String[] args)
0004     {
0005         int n = 2022;
0006         int count = 0;
0007         int tmp = n;
0008         while (tmp != 0)
0009         {
0010             tmp /= 10;
0011             count++;
0012         }
0013         System.out.println(n + "是" + count + "位数");
0014     }
0015
0016 }
```

【运行结果】

程序运行结果如图 3.11 所示。

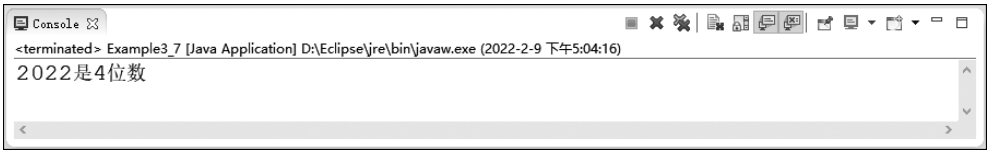

图 3.11　判断自然数位数实例输出

【程序说明】

本程序的功能是求解一个整数的位数。程序第 6 行，初始化工作是设置计数器变量初值为 0；程序第 8 行，循环条件是商不为 0；程序中第 10～11 行为重复的操作，即以 n 不断除以 10，并使用变量 count 计算自然数位数。

3.2.2　do…while 语句

do…while 语句与 while 语句功能类似，其语法格式如下。

```
do
{
    循环体
} while(循环条件);
```

do…while 语句的执行流程如图 3.12 所示。

do…while 语句与 while 语句的区别在于：

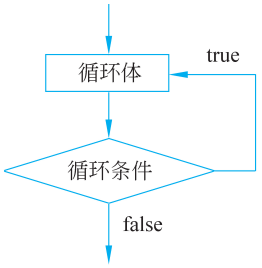

图 3.12　do…while 语句的执行流程

- do…while 先执行循环体,再进行循环条件判断,称为直到型循环结构。
- while 语句是先进行循环条件判断,再执行循环体,称为当型循环结构。


ex3_8

【例 3-8】　do…while 语句实例。

```
0001 public class Example3_8
0002 {
0003     public static void main(String[] args)
0004     {
0005         int m = 21;
0006         int n = 35;
0007         int r;
0008         int t1 = m, t2 = n;
0009         do
0010         {
0011             r = m % n;
0012             m = n;
0013             n = r;
0014         } while(r != 0);
0015         System.out.println(t1 + "和" + t2 + "的最大公约数: " + m);
0016     }
0017 }
```

【运行结果】

程序运行结果如图 3.13 所示。

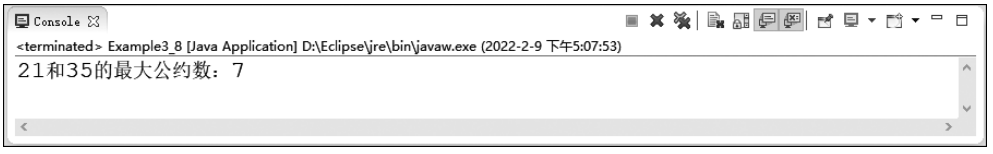

图 3.13　do…while 语句实例输出

【程序说明】

求两个自然数 m、n 的最大公约数应用欧几里得算法,其步骤如下。

第一步:求 m 除以 n 的余数,即 r=m%n。

第二步：用除数和余数分别替换被除数 m 和除数 n，即 m＝n；n＝r。

第三步：判断余数。若余数为 0，则 m 即最大公约数；否则，返回步骤一。

上述例子中，先执行求余数、更新被除数和除数操作，然后再进行条件判断，最后决定是否重复执行相同操作，比较适合使用 do…while 循环。

当然，while 循环和 do…while 循环这两种结构也是可以转换的。如上述例子应用 while 循环同样可以实现，应用 while 循环求最大公约数的主要代码如下。

```
int r = 1;              //先给 r 赋一个不为 0 的值，目的是控制循环进行
while (r != 0)
{
    r = m % n;
    m = n;
    n = r;
}
```

3.2.3　for 语句

当一个循环结构在执行之前可以确定循环次数时，通常使用 for 语句，其语法格式如下。

```
for(初始化赋值语句;  循环条件;  操作表达式)
{
    循环体
}
```

上述语法格式中，for 关键字后面括号()中包含三部分内容：初始化赋值语句、循环条件、操作表达式，它们之间用英文半角分号“;”分隔，后面跟花括号{}，其中的执行语句为循环体。for 循环执行流程如下。

第一步：执行初始化赋值语句。该部分可为空，也可对一个或多个变量赋值。

第二步：判断循环条件。如条件为 true，则执行第三步；否则，执行第五步。

第三步：执行循环体语句。

第四步：执行操作表达式，然后跳转到第二步。

第五步：退出 for 循环。

for 语句的执行流程如图 3.14 所示。

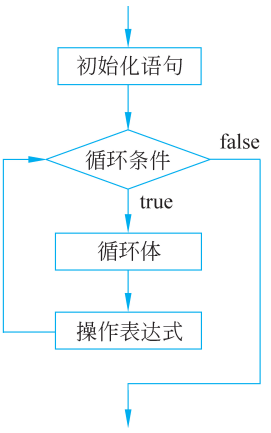

图 3.14　for 语句的执行流程

【例 3-9】　for 语句实例。

```
0001 public class Example3_9
0002 {
```

ex3_9

```
0003    public static void main(String[] args)
0004    {
0005        int i, sum = 0;
0006        for(i = 1; i <= 1000; i++)
0007        {
0008            sum += i;
0009        }
0010        System.out.println("1~1000 的和: " + sum);
0011    }
0012 }
```

【运行结果】

程序运行结果如图 3.15 所示。

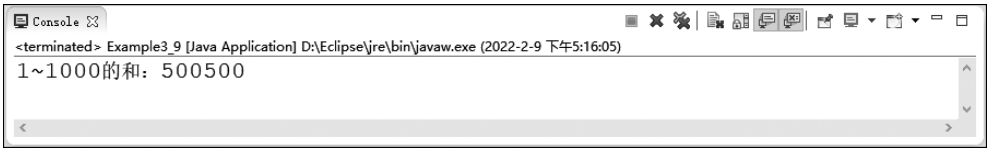

图 3.15　for 语句实例输出

【程序说明】

将本例与例 3-6 比较会发现，for 循环的循环体只有一条累加求和语句，见程序第 8 行，而循环的控制完全地包含在 for 语句中。

for 循环的重要特征是，通常有一个控制循环的变量，称为循环控制变量，它具有初值、终值和变化步长。例如上述例子中，循环控制变量 i 的初值为 1、终值为 1000、步长为 1(即 i＋＋操作，每次增加 1)，该循环执行次数为(终值－初值＋1)÷步长，即循环 1000 次。

3.2.4　break 和 continue 关键字

break 和 continue 是 Java 关键字，也是 Java 中的跳转语句，用于实现循环语句执行过程中程序流程的跳转。

1. break 关键字

在 switch 语句和循环结构中都可以使用 break 语句。在 switch 语句中使用 break 是为了某个 case 执行后即终止并跳出 switch 结构。在循环结构中使用 break 的作用是跳出当前循环，执行循环结构后面的代码。

【例 3-10】　break 语句实例。

ex3_10

```
0001 public class Example3_10
0002 {
0003    public static void main(String[] args)
```

```
0004    {
0005        int n = 25;
0006        int i;
0007        for(i = 2; i <= n - 1; i++)
0008        {
0009            if(n % i == 0)
0010                break;
0011        }
0012        if(i <= n - 1)
0013            System.out.println(n + "不是素数");
0014        else
0015            System.out.println(n + "是素数");
0016    }
0017 }
```

【运行结果】

程序运行结果如图 3.16 所示。

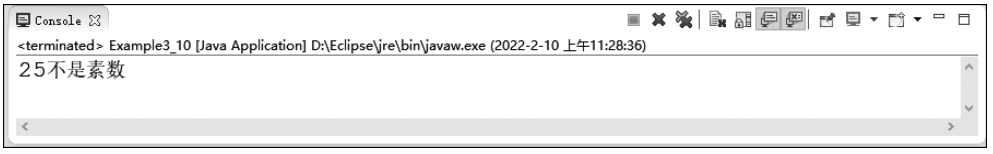

图 3.16　break 语句实例输出

【程序说明】

本程序的功能是判断给定的整数是否为素数，判断的方法是根据素数的定义。对于整数 n，测试[2，n－1]的范围内是否存在能够被 n 整除的数，若有，那么 n 就不是素数；否则，表明 n 是素数。

例 3-10 中，当代码执行到第 12 行时，有两种可能，一种是因为某个 i 使得程序第 9 行条件满足，通过第 10 行 break 语句跳出循环；另一种是 for 循环次数用尽，循环结构自然结束。前者对应 n 不是素数，后者对应 n 是素数。

2. continue 关键字

continue 语句的作用是终止本次循环，继续下一次循环。

【例 3-11】 continue 语句实例。


ex3_11

```
0001 public class Example3_11
0002 {
0003    public static void main(String[] args)
0004    {
0005        int n = 25;
0006        int i;
```

```
0007        for(i = 2; i <= n - 1; i++)
0008        {
0009            if(n % i != 0)
0010                continue;
0011            else
0012            {
0013                System.out.println(n + "不是素数");
0014                return;
0015            }
0016        }
0017        System.out.println(n + "是素数");
0018    }
0019 }
```

【运行结果】

程序运行结果如图 3.17 所示。

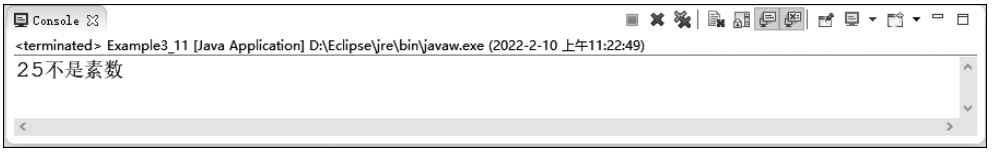

图 3.17　continue 语句实例输出

【程序说明】

本程序的功能与例 3-10 相同,也是判断一个给定的整数是否是素数。程序第 9～10 行,当 n 无法被 i 整除时,并不能说明该数一定是素数,因此需使用 continue 继续除下一个数;程序第 11～15 行,当 n 能够被 i 整除时,说明该数一定不是素数,程序无须继续运行,因此使用 return 语句直接退出方法。

【注意】

- break 的作用是终止循环,跳转到循环后的语句执行。
- continue 的作用是结束本次循环,继续下一次循环。
- break 除了用于循环语句,还用于 switch 语句,contniue 只能用于循环语句。

3.3　数　　组

假设一个班级有 60 人,现在需要处理某个班级某门课程的成绩数据,例如,计算平均分、最高分、分数标准差等,根据前面所学知识,需要定义 60 个变量来存储每位同学的课程成绩,这不仅编写代码麻烦,而且一些操作难以实现。在 Java 中,可以使用一个数组来存储 60 位同学的课程成绩。

3.3.1　Java 数组简介

数组是具有相同类型的一组有序变量的集合,数组中的每个成员称为数组元素,简称

元素。关于数组，需说明的事项如下。

- 数组可以存放各种类型的数据，但同一个数组里存放的元素类型必须一致。
- 数组中元素的个数称为数组的大小，或数组长度。在 Java 中，可通过数组的 length 属性获取数组长度，语法格式为“数组名.length”。
- 数组中每个元素都有一个索引（也称下标），标明其在数组中的位置，要想访问数组中的元素可以通过“数组名[下标]”的方式。
- 数组下标从 0 开始，最大的下标值是“数组长度-1”，访问数组元素时，下标值应在[0，数组长度－1]中，否则会报“ArrayIndexOutOfBoundsException”异常。

根据访问数组元素需要的下标的个数，数组可分为一维数组、二维数组和多维数组，本书介绍一维数组和二维数组。

3.3.2　一维数组

一维数组是指由相同类型元素组成的线性集合，只需使用一个下标就可以访问数组元素。

1. 数组的定义

在 Java 中，定义数组的常用方式有 3 种，其语法格式如下。

方式一：动态初始化，其语法格式如下。

```
数据类型[] 数组名 = new 数据类型[数组长度];
```

或

```
数据类型 数组名[] = new 数据类型[数组长度];
```

方式二：静态初始化，其语法格式如下。

```
数据类型[] 数组名 = new 数据类型[]{元素 0, 元素 1, 元素 2, …};
```

或

```
数据类型 数组名[] = new 数据类型[]{元素 0, 元素 1, 元素 2, …};
```

方式三：静态初始化，其语法格式如下。

```
数据类型[] 数组名 = {元素 0, 元素 1, 元素 2, …};
```

或

```
数据类型 数组名[] = {元素 0, 元素 1, 元素 2, …};
```

其中,数据类型与变量的数据类型一样,数组名即变量名,数组长度表示数组可存放数组元素的个数,元素 0、元素 1 等表示数组中存放的具体数据。

【注意】

- Java 用于声明数组的[]既可在数组名前面,也可放在数组名后面,如 int arr[]和 int[] arr 都是合法的。
- Java 在声明数组时,在[]中不可填入常量表示数组的长度,如 int arr[10]这样的声明在 Java 中是不允许的。
- 若采用方式一定义数组,即不给数组元素进行初始化,则数组中元素的初值为各种数据类型的默认初始值。

注:byte、short、int、long 型元素默认初值为 0,float、double 型元素默认初值为 0.0,char 型元素默认初值为空字符,即'\u0000',boolean 型元素默认初值为 false,引用数据类型默认初值为 null。

下面是一些数组定义示例。

```
int[] a = new int[60];
double scores[] = new double[]{9.8, 9.6, 9.0, 8.6, 9.4};
String[] names = {"alex", "bob", "lily"};
```

方式一定义了 int 类型数组变量 a,并使用关键字 new 申请了连续 60 个 int 型存储单元,分别是 a[0]、a[1]、…、a[59],数组中元素没有初始化,默认初始值均为 0。方式二和方式三定义数组的同时,通过{}完成初始化赋值,这称为静态初始化。

2. 数组的操作

定义数组后,可对其进行各种操作,常见操作包括数组元素赋值、数组遍历、数组求最值、数组排序等。


ex3_12

【例 3-12】 数组遍历实例。

```
0001 public class Example3_12
0002 {
0003     public static void main(String[] args)
0004     {
0005         int[] arr = {1, 3, 5, 7, 9};
0006         arr[2] = 20;
0007         System.out.println("【遍历数组元素】");
0008         for(int i = 0; i < arr.length; i++)
0009             System.out.println("arr[" + i + "]=" + arr[i]);
0010     }
0011 }
```

【运行结果】

程序运行结果如图 3.18 所示。

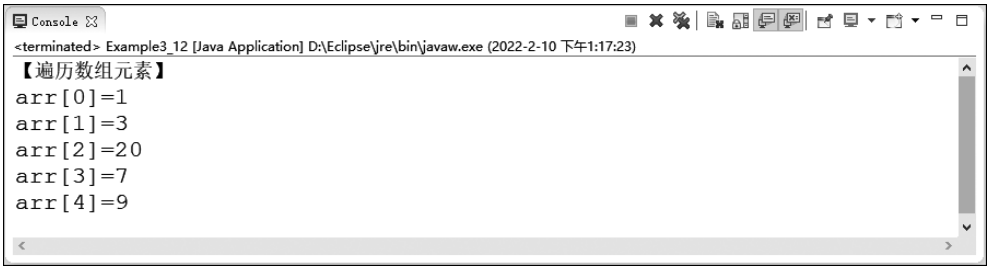

图 3.18　数组遍历实例输出

【程序说明】

本程序的功能是演示数组元素赋值、数组遍历。程序第 5 行定义了整型一维数组，并进行了静态初始化；程序第 7 行给数组元素 arr[2]重新赋值为 20；程序第 9 行通过 for 循环操作一维数组，实现数组遍历，在遍历过程中输出了数组元素的值。

【例 3-13】　数组求最值实例。


```
0001 public class Example3_13
0002 {
0003     public static void main(String[] args)
0004     {
0005         int[] arr = {1, 3, 20, 7, 9, 30, 10};
0006         int max = arr[0];
0007         for(int i = 1; i < arr.length; i++)
0008         {
0009             if(arr[i] > max)
0010                 max = arr[i];
0011         }
0012         System.out.println("数组的最大值为: " + max);
0013     }
0014 }
```

【运行结果】

程序运行结果如图 3.19 所示。

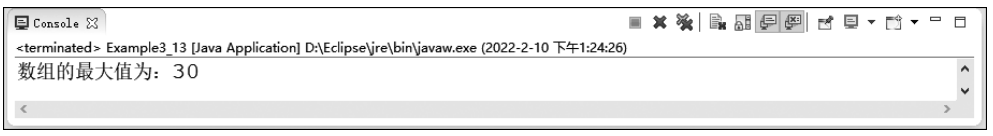

图 3.19　数组求最值实例输出

【程序说明】

本程序使用“擂台法”求数组元素的最值，以求最大值为例。程序第 6 行设置擂台变量 max，用于保存最大值，并初始化其为数组第一个元素 a[0]的值；程序第 7～11 行遍历

数组,将数组元素依次与擂台变量进行比较,如果数组元素的值大于擂台变量的值,则更新擂台变量的值,遍历数组结束后,擂台变量 max 便保存了数组元素的最大值。


ex3_14

【例 3-14】 数组排序实例。

```
0001 public class Example3_14
0002 {
0003     public static void main(String[] args)
0004     {
0005         int[] a = {1, 3, 20, 7, 9, 30, 10};
0006         int n = a.length;
0007         for(int i = 1; i < n; i++)
0008         {
0009             for(int j = 0; j < n - i; j++)
0010             {
0011                 if(a[j] > a[j+1])
0012                 {
0013                     int t = a[j];
0014                     a[j] = a[j+1];
0015                     a[j+1] = t;
0016                 }
0017             }
0018         }
0019         System.out.println("【数组排序后,结果为】");
0020         for(int i = 0; i < n; i++)
0021             System.out.print(a[i] + " ");
0022     }
0023 }
```

【运行结果】

程序运行结果如图 3.20 所示。

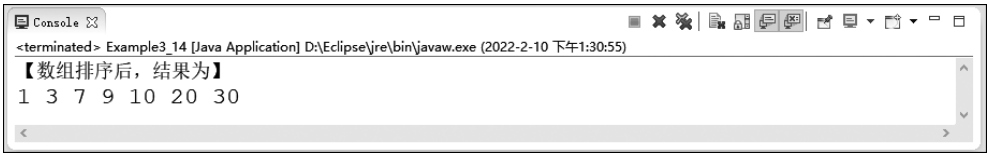

图 3.20 数组排序实例输出

【程序说明】

本程序的功能是对数组进行非降序排序,使用冒泡排序法。冒泡排序是基于交换的排序,排序思想是相邻元素两两比较,如果不满足小数在前、大数在后,则进行交换,这样经过一趟排序后,最大的元素会被交换至数组最后位置。

假设数组有 n 个元素,冒泡排序通过二重 for 循环实现。程序第 7 行代码是外循环,表示 n 个数共进行 n−1 趟排序,每趟排序会产生一个最大的值并交换至无序部分的末

尾，所以 n 个数需要 n－1 趟排序；程序第 9 行代码是内循环，表示第 i 趟排序的过程，对数组元素 a[0]、a[1]、…、a[n－i]进行排序，其方法是遍历此部分数组元素，并比较相邻数组元素，如果大数在前、小数在后则交换二者。

此外，读者可以进一步对上述冒泡排序过程进行改进，例如在某一趟排序后，数组已经有序，则没有必要再进行余下的几趟排序了，提升算法的性能。

3.3.3　二维数组

假设现在需处理某个班级多门课程的成绩数据，每位同学都有多门课程，该如何实现呢？这就需要二维数组，二维数组可以简单地理解为一维数组的数组。

二维数组的定义有 3 种主要方式。

方式一：直接分配数组每一维空间，其语法格式如下。

```
数据类型[][] 数组名 = new 数据类型[数组行数][数组列数];
```

或

```
数据类型 数组名[][] = new 数据类型[数组行数][数组列数];
```

方式二：从数组最高维起，分别为每一维分配空间，其语法格式如下。

第一步：指定二维数组的行数。

```
数据类型[][] 数组名 = new 数据类型[数组行数][];
```

或

```
数据类型 数组名[][] = new 数据类型[数组行数][];
```

第二步：分别指定数组每行的列数。

```
数组名[0] = new 数据类型[数组列数 0];
数组名[1] = new 数组类型[数组列数 1];
…
数组名[数组行数-1] = new 数组类型[数组列数-1];
```

方式二先动态创建数组第一维（即二维数组的行），然后依次为第一维的每个元素分配空间（即每行的列数）。这种方式申请的二维数组的每行列数可不必相同。

方式三：静态初始化，其语法格式如下。

```
数据类型[][] 数组名= {{元素 00, 元素 01, 元素 02, …}, {元素 10, 元素 11, 元素 12},
…, {元素 n0, 元素 n1, 元素 n2, …}};
```

或